PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-134199
(43)Date of publication of application : 21.05.1999

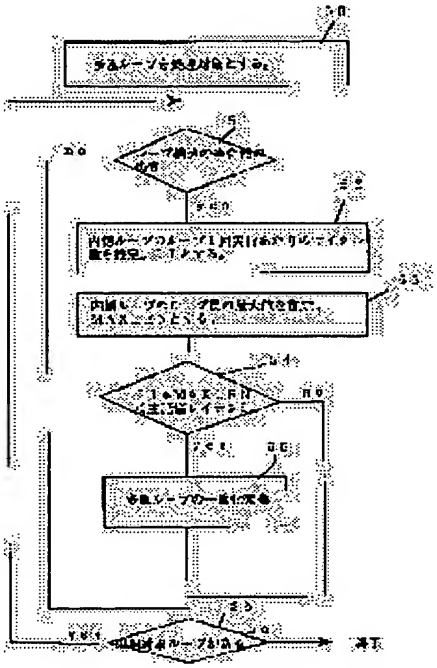
(51)Int.Cl. G06F 9/45
G06F 9/38

(21)Application number : 09-298150
(22)Date of filing : 30.10.1997

(71)Applicant : HITACHI LTD
(72)Inventor : TANAKA GIICHI
TSUSHIMA YUJI
YASAKA SATOSHI

(54) PREFETCH CODE GENERATION SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide an object generation system for hiding main storage penalty when a data area which is accessed by a loop exceeds cache capacity in the multiplex loop.
SOLUTION: A code for obtaining a loop length when the dense multiplex loop part of a source program is expressed by a single loop, a code for storing a value when the control variables of the dense multiplex loop are expressed by the single loops in a one-dimensional array and a code for substituted with the one-dimensional array against the reference of the control variable in the multiplex loop are generated. The single loop code of the dense multiplex loop is generated. A prefetch code transferring data necessary for a repetitive operation in the loop after it is made into the single one to a cache when repetitively processing the loop estimated to be previous by data transfer time.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（Ａ）　(11)特許出願公開番号

特開平11－134199

(43)公開日　平成11年(1999)５月21日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/45		Ｃ０６Ｆ	9/44	３２２Ｃ
	9/38	３３０		9/38	３３０Ｅ

審査請求　未請求　請求項の数４　ＯＬ　（全 11 頁）

(21)出願番号　特願平9－298150

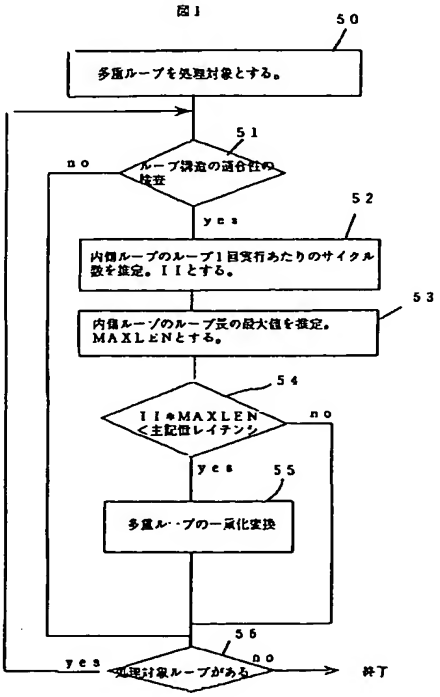
(22)出願日　平成９年(1997)10月30日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地
(72)発明者　田中　義一
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　對馬　雄次
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内
(72)発明者　家坂　聡
神奈川県横浜市戸塚区戸塚町5030番地　株式会社日立製作所ソフトウェア開発本部内
(74)代理人　弁理士　小川　勝男

(54)【発明の名称】　プリフェッチコード生成方式

(57)【要約】
【課題】多重ループにおいて、ループでアクセスするデータ領域がキャッシュ容量を超えた場合に、主記憶ペナルティを隠蔽するオブジェクト生成方式を提供する。
【解決手段】ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を１次元配列に格納するコード、上記多重ループ内の制御変数参照に対し、上記１次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成する。

図1



【特許請求の範囲】
【請求項１】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を１次元配列に格納するコード、上記多重ループ内の制御変数の参照に対し、上記１次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。
【請求項２】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長がコンパイル時に定数として確定する場合は、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を第１の１次元配列に初期値データとして格納し、上記多重ループ内の制御変数の参照に対し、上記１次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。
【請求項３】請求項２において、上記多重ループ内の配列参照に対するベースアドレスに対する相対アドレスがコンパイル時に確定する場合は、上記相対アドレスを一重ループで表現した場合の値を第２の１次元配列に初期値データとして格納し、上記多重ループ内の配列参照の相対アドレス部を上記第２の１次元配列で置換するコードを生成することを特徴とするコード生成方式。
【請求項４】ソースプログラムをオブジェクトプログラムにコンパイルする方式において、上記ソースプログラムの上記密多重ループ部を一重ループで表現した場合のループ長がコンパイル時に定数として確定しない場合、外側ループの二重ループ化を行い、上記二重ループを外側からループ１、ループ２と呼ぶ時にそれぞれのループ長は、最内側ループの処理をループ２だけ繰り返した推定時間が、データ転送時間のコンパイル時の決めた規定倍以上という指針で決め、ループ２と最内側からなる密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を１次元配列に格納するコード、上記多重ループ内の制御変数参照に対し、上記１次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送するプリフェッチコードを生成することを特徴とするコード生成方式。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、命令レベル並列処理を行うプログラムの実行方式に関わり、プログラムがアクセスするデータが大きく、キャッシュに入りきらない場合のループに好適なコード生成方式に関する。
【０００２】
【従来の技術】スーパーコンピュータの一つの方向として、スカラプロセッサをノードプロセッサとする並列処理方式が有望視されている。スカラプロセッサを用いた並列スーパーコンピュータが期待されるのは、半導体技術の進歩によるクロック周波数の向上、複数の並列実行可能な演算器を有効に生かすスーパースカラ方式等の命令レベル並列処理の実現により、スカラプロセッサの処理性能が飛躍的に向上しているためである。
【０００３】しかしながら、その高いスカラプロセッサ処理能力は、キャッシュが有効に働くときのみ達成される。スカラプロセッサは、一般的に命令処理装置とキャッシュ及びメモリ装置を有する。スカラプロセッサはメモリにプログラムとデータを格納し、プログラムに記述された命令に従いメモリ中のデータを処理する。キャッシュは命令処理装置からの参照時間の短い比較的小容量の記憶手段であり、プログラム及びデータの一部を一時的に格納する。
【０００４】命令実行にあたり必要なデータはメモリから読み出されるが、同時にデータを含むデータブロックはキャッシュを構成するラインにコピーされ、以後、当該ブロック内のデータに対する参照が指定されたときは上記キャッシュのラインからデータを参照する。このメモリからキャッシュラインへのデータブロックの転送をライン転送と呼ぶ。命令実行にあたり必要データがキャッシュにない場合、これをキャッシュミスと呼ぶが、キャッシュミスが発生するとライン転送が実行される。
【０００５】従来のスカラプロセッサでは命令実行に伴いこのライン転送が発生すると、その完了まで当該命令の実行が待たされる。従って、キャッシュミスが多発するとライン転送に伴う待ち合わせによりプログラムの実行時間が増大し、スカラプロセッサの処理性能が劣化するという問題があった。特に、大規模科学技術計算では、データ領域が大きくデータの局所性が少ないという性質があるため深刻であった。
【０００６】これに対し、最近では予めプログラムにおいて、キャッシュミスを引き起こす可能性のある命令に先だってデータの先読みを指示する特殊な命令を実行させることで上記ライン転送に伴う性能劣化を回避する試みがなされている。このデータ先読みをプリフェッチと呼ぶ。例えば、米国ＩＢＭ社のマイクロプロセッサPowerPC では指定したオペランドアドレスに位置するデータ

をキャッシュに読み込むプリフェッチ命令がある。

【０００７】この動作において、キャッシュミスを発生した場合、プリフェッチ命令の完了を待ち合わせることなくライン転送を行う。従って、上記データを参照する命令を実行する時には上記データがキャッシュに入っているために上記性能劣化が回避される。このようなプリフェッチ命令を利用して、データ転送と演算をオーバラップさせることで、実質的に主記憶レイテンシを隠蔽させるコード生成技術が特願平9－90470号に示されている。

【０００８】

【発明が解決しようとする課題】上記従来のコード生成技術の基本的考え方は、図７のソースプログラムに示すようなＤＯ２０（１２３）及びＤＯ１０（１２４）の多重ループに対するプリフェッチコードの生成において、外側ループＤＯ２０のｊ＝ｊ１での内側ループＤＯ１０の処理中に、次の外側ループＤＯ２０に関する繰り返しｊ＝ｊ１＋１で内側ループＤＯ１０での処理に必要なデータをプリフェッチすることである。

【０００９】しかし、この方式は内側ループのループ長が小さい時に、主記憶レイテンシが十分に隠蔽できない問題があった。例を用いて具体的に示す。

【００１０】まず、以下のコード生成で前提としたアーキテクチャの仮定を述べる。対象とするスカラプロセッサはロード／ストア命令が２個、浮動小数点演算が２個、及び整数演算が２個、同時に実行でき、キャシュラインサイズが３２バイト、キャッシュミス時の主記憶ペナルティを１００サイクルと仮定する。

【００１１】図７のソースプログラムに対して、コンパイラは、まず、ローカリティ解析などによりプリフェッチ対象オペランドを探す。その結果、参照ｂ(ｉ，ｊ)１２５がプリフェッチ対象オペランドであるとする。オペランドの大きさを８バイトとすると、１回のプリフェッチで、対象とするオペランドを含む連続した３２バイトのデータがキャッシュに格納されるため、ループ繰り返しごとにプリフェッチ命令を実行していたのでは無駄で、この例の場合、４回に１回の割合でプリフェッチ命令を出力すればよい。

【００１２】プリフェッチコードの生成は以下のように行う。まず、ループＤＯ１０のループ１回あたりの実行サイクル数を推定する。ループＤＯ１０では、ロード命令が１個、ストア命令が１個、浮動小数点演算が１個で、プリフェッチ命令が１／４であるので、ループ１回あたりの処理サイクル数は１．１２５サイクル（＝max((１＋１＋１／４）／２，１／２）と推定できる。キャッシュミス時の主記憶ペナルティは１００サイクルであるので、８８回（１００／１．１２５　＝８８）前のループ処理で、現在のループ処理で必要となるデータのプリフェッチを行うコードを生成すると、主記憶ペナルティを隠蔽することができる。

【００１３】しかし、図７の例において、内側ループＤＯ１０のループ長Ｎが８８より小さいときには、従来のコード生成技術では、演算時間が主記憶レイテンシ以下であるので、主記憶レイテンシを完全には隠蔽できない。

【００１４】図３は内側ループ長が６の時の実行の様子を模式的に示したものである。横軸が時間、斜線箱がプリフェッチ処理、空箱が演算処理、箱の上には、内側ループ及び外側ループでの処理ループインデクスｉ，ｊの値を示してある。ただし、図面の都合上、図７のプログラムとは数値的に一致はしていない（ループ１回あたりの実行サイクル数が１．１２５ サイクルでなく、約１０サイクル以上で表示されている）。

【００１５】外側ループｊ＝２及び内側ループｉ＝１の演算処理３２で必要となるデータを、一つ前の外側ループｊ＝１の内側ループｉ＝１での演算処理３０に引き続いてプリフェッチ３１を行っているが、内側ループ長が短く、必要な処理時間が短いため、外側ループｊ＝２及び内側ループｉ＝１の演算処理３２に必要なデータが到着せずストール３３している状態となる。

【００１６】内側ループ長が短い場合のコードの性能を見積もる。ループ長がＮ＝３０の場合、内側ループを実行するのに、演算処理の実行時間は１．１２５×３０＝33.75サイクル、主記憶レイテンシが１００サイクルであるため、内側ループを実行するのに１００サイクルかかり、全体の時間の内６６％（（１００－３３．７５）／１００×１００％）は主記憶待ちのオーバヘッドとなっている。

【００１７】さらに、ループ長が短くＮ＝１０の場合、内側ループを実行するのに、演算処理の実行時間は１．１２５×１０＝１１．２５サイクル、主記憶レイテンシが100サイクルであるため、内側ループを実行するのに１００サイクルかかり、全体の時間の内８８％（（１００－１１．２５）／１００×１００％）は主記憶待ちのオーバヘッドとなっている。

【００１８】本発明の目的は、上記の問題点を解決するプリフェッチ命令を用いたコード生成方式を提供することにある。

【００１９】

【課題を解決するための手段】上記の目的は、ソースプログラムをオブジェクトプログラムにコンパイルする方式において、ソースプログラムの密多重ループ部を一重化ループで表現した場合のループ長を求めるコード、上記密多重ループの各々の制御変数を一重ループで表現した場合の値を１次元配列に格納するコード、上記多重ループ内の制御変数参照に対し、上記１次元配列で置換するコードを生成し、上記密多重ループの一重ループコードを生成し、上記一重化後のループ内のある繰り返しの演算に必要なデータを、データ転送時間だけ前と推定されるループの繰り返し処理時に、キャッシュに転送する

プリフェッチコードを生成することにより達成される。

【００２０】

【発明の実施の形態】以下、本発明のコンパイラにおける実施例を図を参照しつつ説明する。

【００２１】図２にコンパイラ全体の構造を示す。ソースプログラム１が、字句構文解析２によって中間語３に変換される。ループ構造変換部４は、これを入力として、多重ループに関するプリフェッチコードの主記憶レイテンシの隠蔽される割合を高くするための多重ループに関するループ一重化を行い、中間語５を出力する。最適化部６は、通常の公知の伝統的な最適化を行い、中間語７を出力する。コード生成部８は、プリフェッチ命令を含むオブジェクトコード９を、中間語７をもとに生成する。本発明は４及び８に係わり、オブジェクトコード９の実行効率を向上させるものである。

【００２２】図２のループ構造変換部４のうち、多重ループにおける主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図１に示す。図１に入力するソースプログラム１として、図５のＦＯＲＴＲＡＮプログラムを例として説明する。図５のＤＯ２０（７３），ＤＯ１０（７４）はループを示し、これらは二重ループを構成している。このようなプログラムに対して図１は以下のような処理を行い、中間語５に変換する。

【００２３】図１の処理は、ソースプログラム中の多重ループ５０を順次処理する。判定５１は、多重ループを考慮したプリフェッチコードを出力する多重ループを選択する。ここでは、密多重ループに限定する。図５のソースプログラムでは、外側ループＤＯ２０（７３）と内側ループＤＯ１０（７４）の両者に関係する文７５のみが存在するため、この多重ループは密多重ループであるので適合する。

【００２４】この例で、外側ループＤＯ２０（７３）の中に、他のＤＯループがある場合は、本発明では、多重ループを考慮したプリフェッチ命令は出力しないので、ループ構造が適合しないと判断する。処理５２では、元々の内側ループ１回あたりの実行サイクル数を推定する。ループ内には、浮動小数点演算が１個、ロード演算が１個、ストア演算が１個、ラインサイズは３２バイトなので、ループ４回に１個のプリフェッチ命令が出力されると推定されるので、ループ１回あたり、II＝max（１＋１＋１／４）／２）、１／２）＝１．１２５サイクルである。

【００２５】処理５３では、最内側のループ長の最大値を推定する。図７の例では、ループ長は変数Ｎで、コンパイル時には不明であるが、配列宣言１２２から、言語仕様に従った記述であるかぎり、配列の添字は宣言範囲を超えないと考える。この例では、配列ｂの１次元目の宣言から、ループ長Ｎの最大値は５０と判定する。判定５４は、先に求めたループ１回あたりの実行サイクル数と、最大ループ長から、内側ループの処理実行サイクルの最大値を求め、主記憶レイテンシと比較し、主記憶レイテンシが大きい場合は、従来技術では主記憶レイテンシの隠蔽は十分でないと判断し、多重ループの一重化変換５５へ進む。内側ループの処理実行サイクルが主記憶レイテンシより大きい場合は、従来技術を適用するため、一重化変換を行わない。判定５６によりすべての多重ループに対して処理を行う。

【００２６】多重ループの一重化変換５５の詳細な説明は図４を用いて行う。判定６０は一重化後のループ長がコンパイル時に計算可能かを判定する。計算可能な条件は、対象とする多重ループにおいて、外側ループの制御変数の初期値，終値，増分値はコンパイル時に値が確定し（定数）、その他の各ループの制御変数の初期値，終値，増分値は、対象多重ループ内の外側ループの制御変数の関数となることである。図５のＤＯ２０（７３），ＤＯ１０（７４）は、この条件を満たす。これに対し、図７のＤＯ２０（１２３），ＤＯ１０（１２４）は、この条件を満たさない。以下、判定６０が成立したケースを図５の例、成立しないケースを図７の例を用いて説明する。

【００２７】最初に、判定６０が成立する場合を説明する。図５の文７０～文７５は、ソースプログラム１を表しており、字句構文解析部２により、この例の場合、文７６～文８９のようなコンパイラ内部の中間語３に変換される。ここでは、簡単のために、文７９、文８２のようにループに関しては、高級言語のままで表現している。tmp*は整数系の一時変数、ｆtmp*は浮動小数点系の一時変数を意味する。

【００２８】中間語1fd83 は、第２オペランドのアドレス（ｂ（１，１））に、第３オペランドの値（tmp1）を加えた主記憶上のアドレスの８バイト浮動小数点データを第１オペランド（ｆtmp1）にロードする処理、中間語fadd84は、第２オペランド（ｆtmp1）と第３オペランド（ｓ）を浮動小数点加算を行い、第１オペランド（ftmp2）に格納する処理、中間語fst85は、第２オペランドのアドレス（ａ（１，１））に、第３オペランドの値（tmp2）を加えた主記憶上のアドレスに、第１オペランド（ftmp2 ）の８バイトの浮動小数点データを格納する処理、中間語ａｄｄ８６は、第２オペランド（tmp1）と第３オペランド（８）の加算を行い、第１オペランド（tmp1）に格納する処理を表現している。

【００２９】ループ内で、tmp1は配列ｂの原点(ｂ(１，１))、tmp2は配列ａの原点（ａ（１，１））からの相対アドレスを保持している。中間語８６，８７は、最内側ループにおいて、ループ制御変数が１増えるときの、配列ａ，ｂの参照アドレスの増加（８バイト）を表現している。また、中間語８８，８９は、外側ループのループインデクスが１増えた場合の、配列ｂ，配列ａの参照アドレスの増分を表現している。配列ａは、配列宣言７１から、ｋ１×８バイト増加し、配列ｂは、配列宣言７２

から４００バイト増加する。ここで、ｋ１は、コンパイル時にはわからない変数であることに注意を要する。

【００３０】処理６１は一重化後のループ長の計算を行う。これは次のように行えばよい。コンパイラ内部で、各ループのループ制御変数の初期値，増分値，終値から同一ループ構造を作成し、内部に計数用の変数ｃｎｔをおき、実際に計算させればよい。

【００３１】

【数１】cnt＝０
do j＝１，３
do i＝１，j
cnt＝cnt＋１
end do

上記のループで、変数cnt に一重化後のループ長を求めることができ、この例の場合、６となる。

【００３２】処理６２は、制御変数配列への初期値代入を行う。これは、ループ制御変数を、上記の一重ループで表現したときの値であり、上記の一重ループ化後のループ長を求める時と同様に、次のように求めることができる。

【００３３】

【数２】cnt＝０
do j＝１，３
do i＝１，j
¥IL（cnt＋１）＝i
¥JL（cnt＋１）＝j
cnt＝cnt＋１
end do

ここで、¥ILが内側ループ制御変数の値であり、¥JLは外側ループ制御変数の値である。これを、初期値付データとして扱う。図６の９１／９２が、求められた初期値付データを表現している。なお、本例では、ループ制御変数を直接的に使うことはなく、８バイトデータの配列のアドレスとしてしか用いられていないので、８倍した値を設定している。

【００３４】処理６３は、配列のベースアドレスよりの相対アドレスが確定する参照に対して、その値を相対アドレス配列への初期値として代入する。相対アドレスが確定する参照とは、配列ｂの参照７５のように一重化後のループインデクスの値に対して、配列のベースアドレス（ｂ（１，１））からの相対アドレスが定数で決まる場合をいう。配列ｂ（ｉ，ｊ）の相対アドレスは、ｂ（ｉ，ｊ）のアドレスとｂ（１，１）のアドレスの差が（ｉ－１＋５０×（ｊ－１））×８バイトであることから

【００３５】

【数３】cnt＝０
do j＝１，３
do i＝１，j
¥ｂｉｎｄ（cnt＋１）＝（ｉ－１＋５０＊（ｊ－１））＊８
cnt＝cnt＋１
end do

というプログラムにより求めることができる。ここで、¥bindが参照ｂ（ｉ，ｊ）の相対アドレスデータであり、図６の９０が、求められた初期値データを表している。なお、配列参照ａ（ｉ，ｊ）は、相対アドレスが（ｉ－１＋ｋ１＊（ｊ－１））＊８ であるが、変数としてループ制御変数ｉ，ｊ以外のｋ１（配列ａの宣言７１の１次元目）を含んでいるために、コンパイル時に定数として計算できない。これに関しては実行時に計算することになる。

【００３６】処理６４は多重ループ部分のループ一重化変換を行う。ＤＯ２０（７９）とＤＯ１０(８２)がさきほど求めたループ長が６となる一重ループＤＯ３０(９７)に置き換わる。配列参照ｂ（ｉ，ｊ）のアドレス計算コードは、７７，８０，８６，８８から構成されていたが、一重化により、コード９４，９８，１０６に置き換わる。ここでは、配列参照ｂ（ｉ，ｊ）は、処理６５により、相対アドレス計算コードが除去され、先に求めた相対アドレス配列¥bindをロード（９８）する処理に最適化されている。

【００３７】また、配列参照ａ（ｉ，ｊ）のアドレス計算コードは、７６，７８，８１，８７，８９から構成されていたが、一重化により、コード９３，９５，９６，１０１，１０２，１０３，１０４，１０７，１０８に置き換わる。ここで、コード１０１，１０７が、制御変数¥ILの値、コード１０２，１０８が、制御変数¥JLの値を意味している。配列参照ｂ(ｉ，ｊ)の相対アドレス（ｉ－１＋ｋ１＊(ｊ－１))は、これらから、コード１０３，１０４から求めることができる。

【００３８】次に、判定６０が成立せず、一重化後のループ長がコンパイル時に計算できない場合に関して、図７を用いて説明する。図５のプログラムとの違いは、ループインデクスの範囲（１２３，１２４）が異なり、コンパイル時に一重化後のループ長を求めることができないことである。このため、図５の処理においてコンパイル時に行っていた操作を、コードとして生成し、実行時に求めることになる。

【００３９】まず、処理６６において外側ループを二つの多重ループ（外側ループ１と外側ループ２）に分解する。分割の基準は、外側ループ２と内側ループを一重化した場合の処理時間と主記憶レイテンシの時間比を例えば２０程度とするように決める。この値を大きくすればするほど、主記憶レイテンシのオーバヘッドが見えなくなるが、一重化のための作業配列容量が大きくなる。

【００４０】図７の例の場合、ＤＯ２０（１２９）は、図８のＤＯ２５（１５１），ＤＯ２６（１５４）の二つに分けられる。ここで、コード１４５～１５０及び、コード１５２～１５３，１５５は二重ループ化のために必要なものである。ここでの分割点は次のように求めた。

まず、ループ１回あたりの実行サイクル数を推定する。

【００４１】コード１３３～１３７の必要な演算資源は、メモリ系命令が２、浮動小数点系命令が１、整数系命令が２であるが、一重化されることによりメモリ参照系命令のアドレス計算コードが増加し、さらに配列参照ｂに対するプリフェッチコードのためにコードが増加し、結局、メモリ系命令が７、浮動小数点系命令が１、整数系が８となる（図１１のコードＤＯ４２）と推定される。メモリ系命令が２個、浮動小数点系命令が２個、そして整数系命令が２個が同時に実行できる計算機の場合、このループの１回あたりの実行サイクル数は４（＝max(７／２，１／２，８／２)）と推定できる。また、最内側ループ長Ｎ（１２４）の最大値は、配列宣言１２２より５０と推定されるので、最内側ループの処理時間の最大値は２００（＝４×５０）サイクルと推定される。主記憶レイテンシを１００サイクルと仮定し、主記憶レイテンシのオーバヘッドを１／２０とするには最内側処理を約１０回まわすように外側ループ２を決めればよい。

【００４２】処理６７，処理６８は、外側ループ２と内側ループの一重化後のループ長計算中間語、及び、制御変数配列への初期値代入中間語をコード中に挿入する。図９のコード１６０～１６６がこれを意味する。コード１６７のtmp28 に一重化後のループ長が保持される。また、ここで、制御変数配列が新たに導入されたが、コード１４４のように、最大ループ長５００要素分（５０（内側ループ最大ループ長）×１０（外側ループ２の最大ループ長））を確保する。

【００４３】処理６９は、外側ループ２と内側ループのループ一重化による中間語変換を行う。図８のＤＯ２６（１５４）とＤＯ１０（１３２）の含む文のループの一重化を行うことにより、図９のＤＯ４０（１７０）のループ一重化が行われる。配列参照ｂ（ｉ，ｊ）のアドレス計算コードは、１３０，１３６，１３８から構成されていたが、一重化により、コード１６８，１６９，１７１，１７２，１７３，１７４，１８０，１８１に置き換わる。また配列参照ａ（ｉ，ｊ）のアドレス計算コードは、１３１，１３７，１３９から構成されていたが、一重化により、コード１６８，１６９，１７１，１７２，１７７，１７８，１８０，１８１に置き換わる。両参照に共通であるコード１７１，１７２，１８０，１８１は、元々のループでの制御変数配列の値のロード処理を意味する。

【００４４】その後、通常の中間語に対する最適化部６の処理を行い、コード生成部８においてコード生成を行う。コード生成部８における、多重ループの一重化による主記憶アクセスペナルティの効率的な隠蔽に関わる部分を図１０に示す。処理200は、最内側ループ内のプリフェッチ対象ロードオペランドの選択を行う。選択の方法は、例えば、公知技術であるTodd C.Mowry Monica S. Lam Anoop Gupta Design and Evaluation of a Compiler Algorithm for Prefetching,ASPLOS－V,ACM 0－89791－535－6／92／0010／0062 pp.62－73に従う。この結果、図９のｂ（ｉ，ｊ）１７５がプリフェッチ対象オペランドとなるとする。

【００４５】処理２０１では、最内側ループのループ範囲を二つに分割する。後半ループのループ長には、実行時間を主記憶アクセスペナルティ以上にするループ長 、か、または、ループ長が短く主記憶アクセスペナルティ以上の実行時間にならないときは、もともとのループ長を設定する。前半ループのループ長はその残りとする。従って、前半ループは実行されないこともある。図９のＤＯ４０（１７０）のループ１回あたりの実行推定サイクルは前に述べたようにＭＩＮＬ＝４である。このため、主記憶レイテンシを１００とすると、後半ループのループ長は、tmp50＝min(tmp28，１００／４)となる。その結果、図９のＤＯ４０（１７０）は、図１１のループ長tmp51(２２３）の前半ループＤＯ４２（２３２）、図１２のループ長tmp50（２２２）の後半ループＤＯ４３（２５１）に分割できる。

【００４６】処理２０２は、前半ループで必要となるプリフェッチコード生成を行う。図１１のＤＯ４１（２２４）が対応するコードである。これは、配列参照ｂ（ｉ，ｊ）のアドレス計算コード１７１，１７２，１７３，１７４，１８０，１８１をコピーして一時変数をリネームすれば得られる。すなわち、tmp35をtmp65，tmp36をtmp66，tmp37をtmp67，tmp38をtmp68，tmp30をtmp80，tmp31をtmp81にリネームすればよい。後者の番号を用いたコード系列がプリフェッチを生成するためのアドレス計算を意味する。コード２２０，コード２２１は、リネームした一時変数の初期値の設定である。このループのループ長は、tmp50 で、上記の二分割ループの後半のループと同一である。また、prefetch229 は、第１オペランド（ｂ(１，１)）のアドレスに第２オペランド（tmp38）を加算したアドレスを含む１ラインのデータを主記憶からキャッシュに転送する中間語を意味する。

【００４７】処理２０３は、上記前半ループのコード生成を行う。ここでは、もともとのループボディ（コード１７１～１８１）に、処理２０１と同一のプリフェッチのためのコード（２３３～２３７，２３８，２３９）を挿入したものである。これは、ループｉ回目において、ループ（ｉ＋１００／４）回で必要となるデータをプリフェッチしていることになる。

【００４８】処理２０４は、上記の後半ループに対するコード生成を行う。コード１７１～１８１とループ長を除いてまったく同一である。

【００４９】そして、実際の機械語を生成し、オブジェクトコード９を生成する。

【００５０】

【発明の効果】本発明のコード生成方式により、多重ル

ープにおいて、ループを一重化して、必要なデータを主記憶レイテンシの時間に相当するループ繰り返し前にプリフェッチを開始するため、主記憶レイテンシを隠蔽することができる。図７の例で内側ループＮが１０で、外側ループ長Ｍが１０の時の処理時間を推定する。従来方式では、先に述べたように、内側ループ長が小さいときには、図3のように、内側ループの処理の完了毎にデータ待ちのストールが発生するため、１０００サイクル（１００（内側ループ処理時間）×１０（外側ループ長））となる。

【００５１】本方式によれば、内側ループと外側ループの全体の長さが長ければ、図１３のように、主記憶ペナルティは最初を除き軽減することができる。このため、本方式のコードではループ１回あたり４サイクルの時間がかかることから、ループ処理での４００＝４×１０（内側ループ長）×１０（外側ループ長）とループ最初のデータの待ちの１００サイクルのオーバヘッドを合わせて５００サイクルとなる。この性能は従来方式に比べて約２倍の性能である。

【図面の簡単な説明】

【図１】本発明による多重ループ処理方式を示すフロー図。

【図２】本発明のコンパイラの全体の構成と機能を示す説明図。

【図３】従来法による実行の様子を示した説明図。

【図４】本発明によるループ一重化方式のフロー図。

【図５】本発明による多重ループの一例のプログラムを示す図。

【図６】本発明による多重ループの一重化変換の一例のプログラムを示す図。

【図７】本発明による多重ループの他の例のプログラムを示す図。

【図８】本発明による多重ループの一重化変換の他の例のプログラムを示す図。

【図９】本発明による多重ループの一重化変換の他の例のプログラムを示す図。

【図１０】本発明によるプリフェッチコード生成方式を示すフロー図。

【図１１】本発明による多重ループのプリフェッチ変換の一例のプログラムを示す図。

【図１２】本発明による多重ループのプリフェッチ変換の他の例のプログラムを示す図。

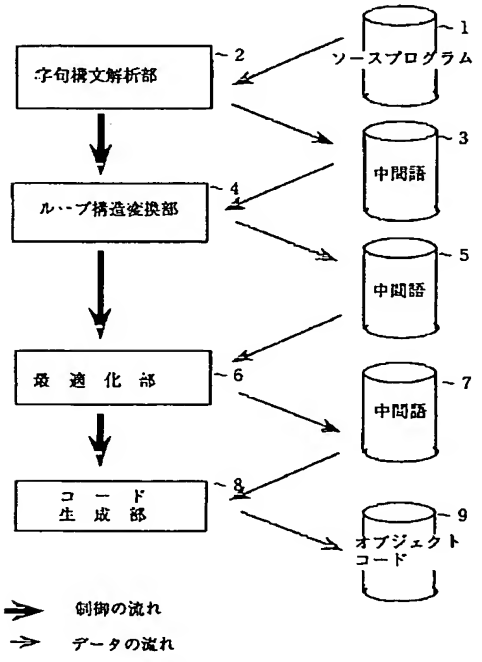
【図１３】本発明による実行の様子を示した説明図。

【符号の説明】

1…ソースプログラム、３…字句構文解析、３…中間語、４…ループ構造変換部、５…中間語、６…最適化部、７…中間語、８…コード生成部、９…プリフェッチ命令を含むオブジェクトコード。

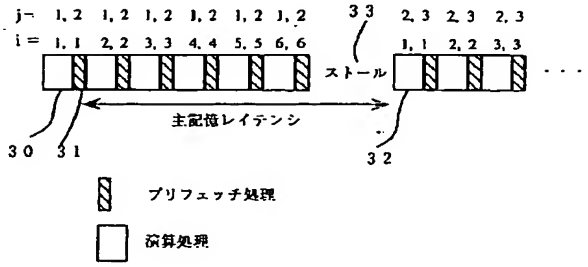
【図２】

図２



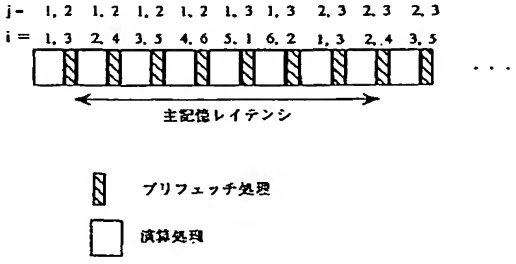
【図３】

図３



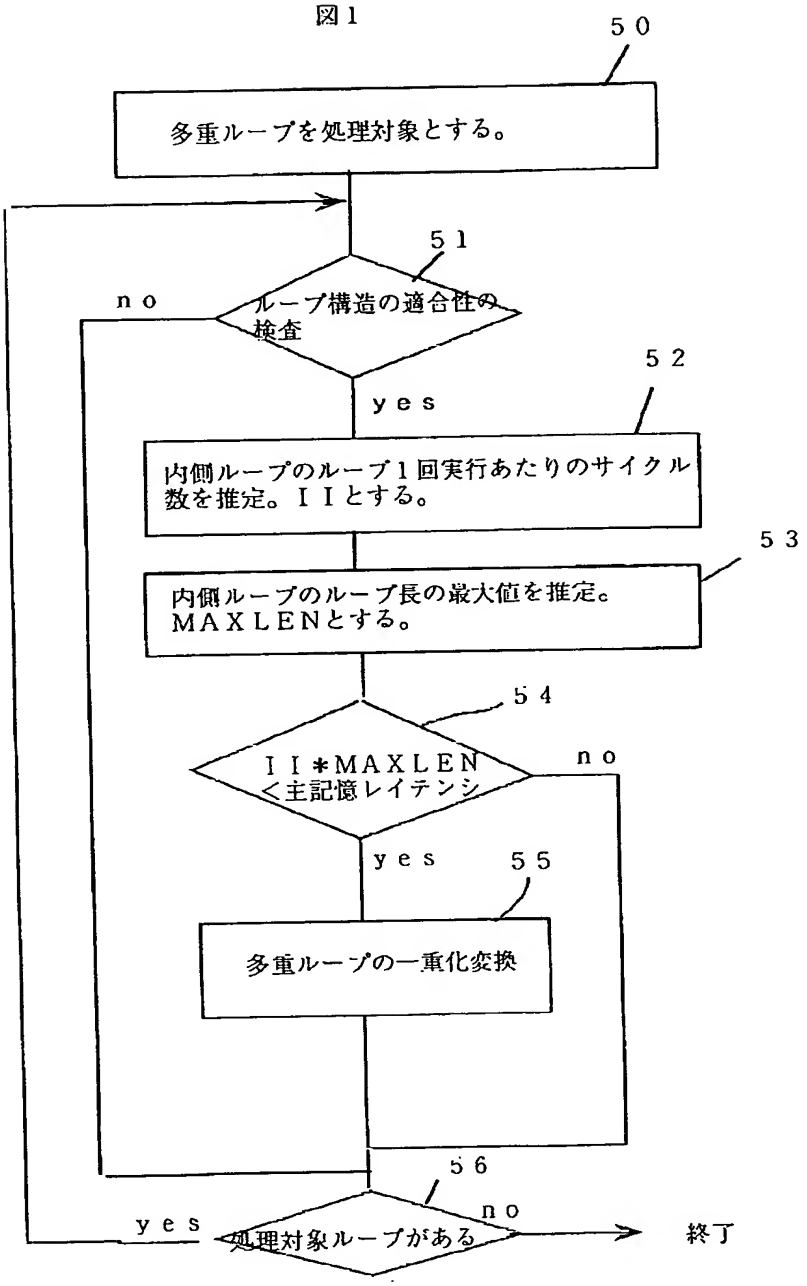
【図１３】

図１３



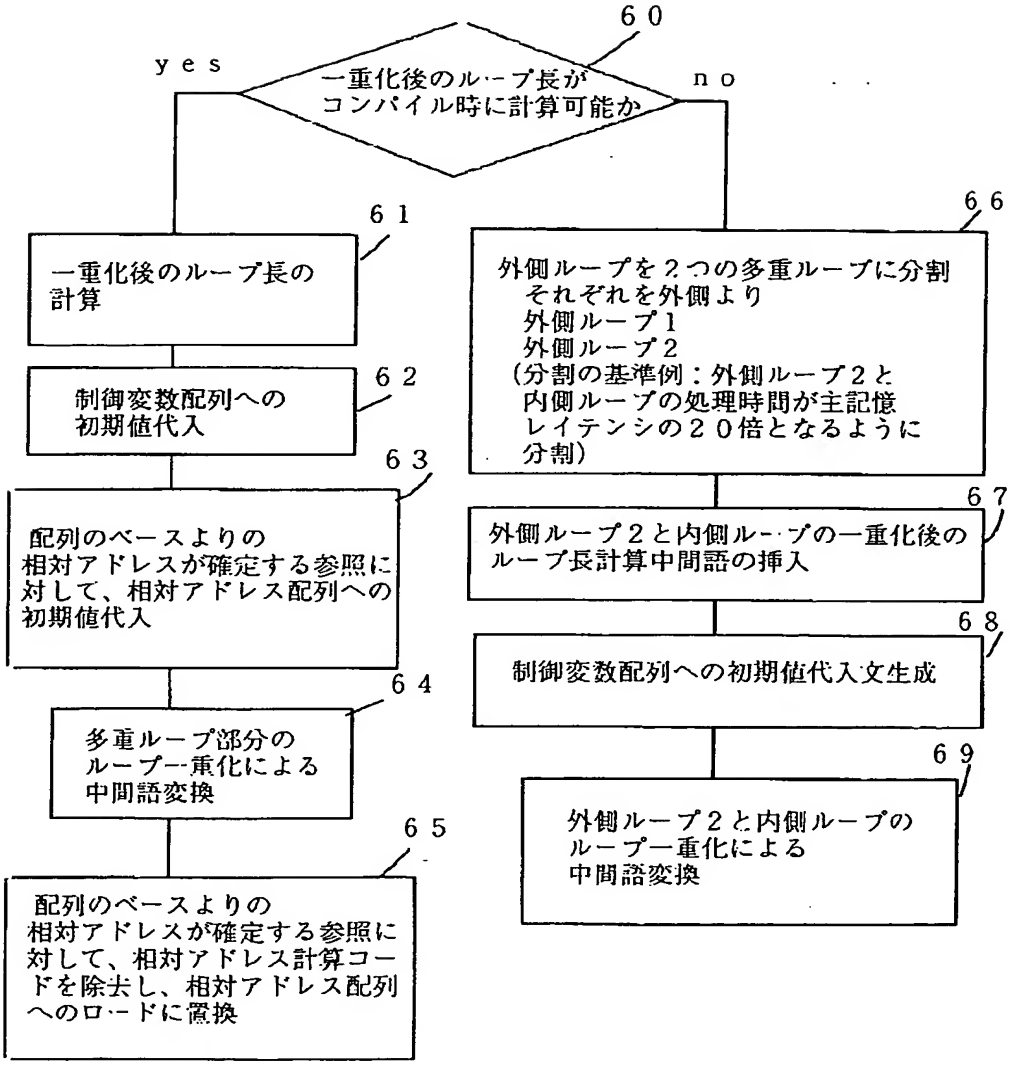
【図１】

図１


５０
多重ループを処理対象とする。
５１
ループ構造の適合性の検査
ｎｏ
ｙｅｓ
５２
内側ループのループ１回実行あたりのサイクル数を推定。ＩＩとする。
５３
内側ループのループ長の最大値を推定。ＭＡＸＬＥＮとする。
５４
ＩＩ＊ＭＡＸＬＥＮ＜主記憶レイテンシ
ｎｏ
ｙｅｓ
５５
多重ループの一重化変換
５６
ｙｅｓ
処理対象ループがある
ｎｏ
終了

【図４】

図４


６０
一重化後のループ長がコンパイル時に計算可能か
ｙｅｓ
ｎｏ
６１
一重化後のループ長の計算
６２
制御変数配列への初期値代入
６３
配列のベースよりの相対アドレスが確定する参照に対して、相対アドレス配列への初期値代入
６４
多重ループ部分のループ一重化による中間語変換
６５
配列のベースよりの相対アドレスが確定する参照に対して、相対アドレス計算コードを除去し、相対アドレス配列へのロードに置換
６６
外側ループを２つの多重ループに分割
それぞれを外側より
外側ループ１
外側ループ２
（分割の基準例：外側ループ２と内側ループの処理時間が主記憶レイテンシの２０倍となるように分割）
６７
外側ループ２と内側ループの一重化後のループ長計算中間語の挿入
６８
制御変数配列への初期値代入文生成
６９
外側ループ２と内側ループのループ一重化による中間語変換

【図5】

図5

```
      subroutine xxx(a, kl)          —70
      real*8 a(kl, 100)              —71
      real*8 b(50, 100)              —72

      DO 20 j=1, 3                   —73
      DO 10 i=1, j                   —74

        a(i, j)=b(i, j)+s            —75

10    continue
20    continue
```

```
      tmp5=8*kl                      —76
      tmp3=0                         —77
      tmp4=0                         —78

      DO 20 j=1, 3                   —79

        tmp1=tmp3                    —80
        tmp2=tmp4                    —81

      DO 10 i=1, j                   —82

        lfd   ftmp1, b(1, 1), tmp1   —83
        fadd  ftmp2, ftmp1, s        —84
        fst   ftmp2, a(1, 1), tmp2   —85

        add   tmp1, tmp1, =8         —86
        add   tmp2, tmp2, =8         —87

10    continue

        add   tmp3, tmp3, =8*50      —88
        add   tmp4, tmp4, tmp5       —89

20    continue
```

【図7】

図7

```
      subroutine xxx(a, kl)          —120
      real*8 a(kl, 100)              —121
      real*8 b(50, 100)              —122

      DO 20 j=1, M                   —123
      DO 10 i=1, N                   —124

        a(i, j)=b(i, j)+s            —125

10    continue
20    continue
```

```
      tmp5=8*kl                      —126
      tmp3=0                         —127
      tmp4=0                         —128

      DO 20 j=1, M                   —129

        tmp1=tmp3                    —130
        tmp2=tmp4                    —131

      DO 10 i=1, N                   —132

        lfd   ftmp1, b(1, 1), tmp1   —133
        fadd  ftmp2, ftmp1, s        —134
        fst   ftmp2, a(1, 1), tmp2   —135

        add   tmp1, tmp1, =8         —136
        add   tmp2, tmp2, =8         —137

10    continue

        add   tmp3, tmp3, =8*50      —138
        add   tmp4, tmp4, tmp5       —139

20    continue
```

【図6】

図6

```
主記憶領域
  ¥bind /0, 400, 408, 800, 808, 816/—90
  ¥IL  /8, 8, 16, 8, 16, 24/         —91
  ¥jL  /8, 16, 16, 24, 24, 24/       —92


      tmp13=kl                       —93
      tmp7=0                         —94
      tmp8=0                         —95
      tmp9=0                         —96

    DO 30 ii=1, 6                    —97

      l     tmp6, ¥bind(1), tmp7     —98
      lfd   ftmp1, b(1, 1), tmp6     —99

      fadd  ftmp2, ftmp1, s          —100

      l     tmp10, ¥IL(1), tmp8      —101
      l     tmp11, ¥jL(1), tmp9      —102
      mul   tmp12, tmp11, tmp13      —103
      add   tmp14, tmp10, tmp12      —104
      fst   ftmp2, a(1, 1), tmp14    —105

      add   tmp7, tmp7, =4           —106
      add   tmp8, tmp8, =4           —107
      add   tmp9, tmp9, =4           —108

30    continue
```

【図8】

図8

```
      tmp5=8*kl                      —126
      tmp3=0                         —127
      tmp4=0                         —128

      tmp20=10                       —145
      tmp25=tmp20                    —146
      tmp21=M/tmp20                  —147
      tmp22=M-tmp21*tmp20            —148
      if(tmp22. ne. 0) then          —149
        tmp21=tmp21+1                —150
      endif

    DO 25 j1=1, tmp21                —151

      if(j1. eq. tmp21. and.         —152
         tmp22. ne. 0) then
        tmp25=tmp22                  —153
      end if

    DO 26 j2=1, tmp25                —154

      j=j2+tmp20*(j1-1)              —155

      tmp1=tmp3                      —130
      tmp2=tmp4                      —131

    DO 10 i=1, N                     —132

      lfd   ftmp1, b(1, 1), tmp1     —133
      fadd  ftmp2, ftmp1, s          —134
      fst   ftmp2, a(1, 1), tmp2     —135

      add   tmp1, tmp1, =8           —136
      add   tmp2, tmp2, =8           —137

10    continue

      add   tmp3, tmp3, =8*50        —138
      add   tmp4, tmp4, tmp5         —139

26    continue
25    continue
```

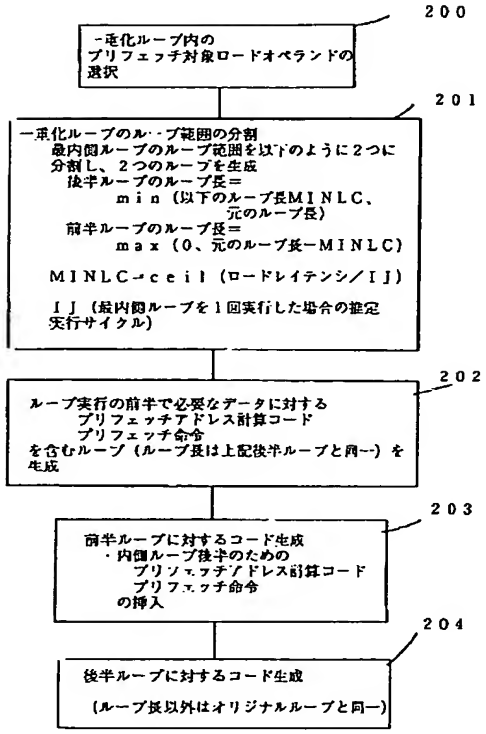
【図9】

図9

```
Dimension ¥IL(500),¥JL(500)          ―144

      tmp20=10                       ―145
      tmp25=tmp20                    ―146
      tmp21=M/tmp20                  ―147
      tmp22=M-tmp21*tmp20            ―148
      if (tmp22.ne.0) then           ―149
        tmp21=tmp21+1                ―150
      endif

      DO 25 j1=1,tmp21               ―151

      if (j1.eq.tmp21.and.           ―152
          tmp22.ne.0) then
        tmp25=tmp22                  ―153
      end if

      tmp27=1                        ―160
      DO 30 j2=1,tmp25               ―161
      DO 31 i=1,N                    ―162
        j=j2+tmp20*(j1-1)            ―163
        ¥IL(tmp27)=i*8               ―164
        ¥JL(tmp27)=j*8               ―165
        tmp27=tmp27+1                ―166
31 continue
30 continue

      tmp28=tmp27-1                  ―167
      tmp30=0                        ―168
      tmp31=0                        ―169

      DO 40 ii=1,tmp28               ―170
        l     tmp35,¥IL(1),tmp30     ―171
        l     tmp36,¥JL(1),tmp31     ―172
        mul   tmp37,tmp36,=50        ―173
        add   tmp38,tmp35,tmp37      ―174
        lfd   ftmp10,b(1,1),tmp38    ―175
        fadd  ftmp11,ftmp10,s        ―176
        mul   tmp38,tmp36,k1         ―177
        add   tmp39,tmp35,tmp38      ―178
        fst   ftmp11,a(1,1),tmp39    ―179
        add   tmp30,tmp30,=4         ―180
        add   tmp31,tmp31,=4         ―181

40 continue

25 continue
```

【図10】

図10



【図11】

図11

```
      tmp30=0                        ―168
      tmp31=0                        ―169
      tmp80=tmp30                    -220
      tmp81=tmp31                    ―221

      tmp50=min(100/4,tmp28)         ―222
      tmp51=tmp28-tmp50              -223

      DO 41 ii=1,tmp50               ―224
        l     tmp65,¥IL(1),tmp80     -225
        l     tmp66,¥JL(1),tmp81     -226
        mul   tmp67,tmp66,=50        -227
        add   tmp68,tmp65,tmp67      -228
        prefetch b(1,1),tmp38        ―229
        add   tmp80,tmp80,=4         -230
        add   tmp81,tmp81,=4         -231
41 continue

      DO 42 ii=1,tmp51               ―232
        l     tmp65,¥IL(1),tmp80     -233
        l     tmp66,¥JL(1),tmp81     -234
        mul   tmp67,tmp66,=50        ―235
        add   tmp68,tmp65,tmp67      -236
        prefetch b(1,1),tmp38        ―237
        l     tmp35,¥IL(1),tmp30     ―171
        l     tmp36,¥JL(1),tmp31     ―172
        mul   tmp37,tmp36,=50        ―173
        add   tmp38,tmp35,tmp37      ―174
        lfd   ftmp10,b(1,1),tmp38    ―175
        fadd  ftmp11,ftmp10,s        ―176
        mul   tmp38,tmp36,k1         ―177
        add   tmp39,tmp35,tmp38      ―178
        fst   ftmp11,a(1,1),tmp39    ―179
        add   tmp30,tmp30,=4         -180
        add   tmp31,tmp31,=4         ―181
        add   tmp80,tmp80,=4         ―238
        add   tmp81,tmp81,=4         ―239
42 continue
```

【図12】

図12

```
   DO 43 ii=1,tmp50                  ―251

     l     tmp35,¥IL(1),tmp30       ―252
     l     tmp36,¥JL(1),tmp31       ―253
     mul   tmp37,tmp36,=50          ―254
     add   tmp38,tmp35,tmp37        ―255
     lfd   ftmp10,b(1,1),tmp38      ―256
     fadd  ftmp11,ftmp10,s          ―257
     mul   tmp38,tmp36,k1           ―258
     add   tmp39,tmp35,tmp38        ―259
     fst   ftmp11,a(1,1),tmp39      ―260
     add   tmp30,tmp30,=4           ―261
     add   tmp31,tmp31,=4           ―262

43 continue
```